Edwin Jones

CS 405: Secure Coding

June 7, 2025

Process Summary

Cppcheck and clang-tidy were used to perform static analysis on the source code. These tools provided overlapping and distinct results, allowing for a comprehensive review of code quality and security issues.

The following commands were used to run the analysis:

**Cppcheck:**  
cppcheck --enable=all --inconclusive --suppress=missingIncludeSystem --std=c++17 --xml --xml-version=2 --output-file=cppcheck-result.xml QuestionableCode.cpp

**clang-tidy:**  
clang-tidy QuestionableCode.cpp -- -std=c++17 2>&1 > clang-tidy-result.txt

These configurations enabled detection of logic errors, undefined behavior, and code quality concerns, while explaining why some system-level or build-specific warnings may not appear in the results.

Cppcheck and clang-tidy both flagged issues with assert(my\_function() == 3), but for different reasons. Cppcheck noted that the assertion includes a side-effecting function call and compares a boolean to an integer. Clang-tidy reported it as a tautological comparison. To fix this, avoid side effects in assertions and compare boolean expressions to true or false only.

In the foo() function, both tools flagged a high-risk dangling pointer issue. The address of a local stack variable is assigned to a pointer argument, which becomes invalid after the function ends. This should be fixed by avoiding the return of local variable addresses. If necessary, dynamic memory allocation can be used to safely preserve the data.

Cppcheck uniquely reported a high-risk out-of-bounds write in work\_with\_arrays(), where an array of size 10 is accessed at index 1000. This type of issue can result in memory corruption and should be mitigated by validating the index before accessing the array. This issue was not reported by clang-tidy.

Cppcheck also identified a separate assertion misuse where assert(z = 2) performs an assignment rather than a comparison. This is a medium-risk issue that should be corrected by replacing the assignment with a comparison, such as assert(z == 2).

Clang-tidy flagged a dead store on a variable that is assigned a value but never read before being overwritten. This is a low-risk code quality issue. The variable should either be used appropriately or the assignment should be removed. Cppcheck did not report this.

A high-risk potential null pointer dereference was identified by both tools in a function that calls tok->next() without checking if tok is null. To fix this, a null check should be performed before the method is called.

Cppcheck identified that my\_function() returns a non-boolean value from a function declared to return bool. This medium-risk issue could lead to incorrect logic in dependent code. The function should be updated to return an explicit true or false value.

Variable shadowing was detected by Cppcheck in the try block within main(), where inner variables x, y, and z shadow outer ones. Although this is a low-risk issue, it reduces code clarity and should be resolved by renaming the inner variables.

Cppcheck also flagged a medium-risk issue in vector\_test(), where an element is erased from a vector during iteration without updating the iterator correctly. This can cause undefined behavior. The fix is to use an iterator-safe erase pattern, such as reassigning the iterator after erasure.

Additional findings from Cppcheck included unused functions and uninitialized class members. These are not security vulnerabilities but reflect poor maintainability and should be cleaned up to improve code quality.

Overall, Cppcheck was more effective in identifying memory safety issues and common C/C++ logic errors, such as buffer overflows and pointer misuse. Clang-tidy was more focused on C++ language correctness and structure, contributing findings related to dead stores and misuse of modern C++ features. Using both tools together resulted in a more thorough and accurate analysis than relying on either tool alone.